Desenvolvedor de back-end em 30 dias: Adquira habilidades em design de API, gerenciamento de dados, teste de aplicativos, implantação, segurança e otimização de desempenho

Processo interno
Desenvolvedor em 30 dias

Adquira habilidades em design de API, Gerenciamento de dados, teste de aplicativos, Implantação, segurança e otimização de desempenho

---

Por Jideon F. Marques

© Copyright 2022 Jideon Marques - Todos os direitos reservados.

O conteúdo deste ebook não pode ser reproduzido, duplicado ou transmitido sem permissão direta por escrito do autor ou do editor.

Sob nenhuma circunstância qualquer culpa ou responsabilidade legal será imputada ao editor, ou autor, por quaisquer danos, reparações ou perdas monetárias devido às informações contidas neste ebook, direta ou indiretamente.

Notícia legal:

Este ebook é protegido por direitos autorais. É apenas para uso pessoal. Você não pode alterar, distribuir, vender, usar, citar ou parafrasear qualquer parte ou o conteúdo deste ebook sem o consentimento do autor ou editor.

Aviso de isenção de responsabilidade:

Observe que as informações contidas neste documento são apenas para fins educacionais e de entretenimento. Todo esforço foi feito para apresentar informações precisas, atualizadas, confiáveis e completas. Nenhuma garantia de qualquer tipo é declarada ou implícita. Os leitores reconhecem que o autor não está envolvido na prestação de aconselhamento jurídico, financeiro, médico ou profissional. O conteúdo deste ebook foi derivado de várias fontes. Consulte um profissional médico licenciado antes de tentar este programa ou qualquer técnica descrita neste ebook.

Ao ler este documento, o leitor concorda que em nenhuma circunstância o autor é responsável por quaisquer lesões, morte, perdas, diretas ou indiretas, que sejam incorridas como resultado do uso das informações contidas neste documento, incluindo, mas não limitado a a, erros, omissões ou imprecisões.

Prefácio

Este livro aborda o processo de construção de aplicativos de software em larga escala do ponto de vista de um desenvolvedor de back-end. Mais do que um tutorial sobre ferramentas ou estruturas específicas, este livro descreve princípios e processos compartilhados entre as várias pilhas de tecnologia. Ferramentas e frameworks mudam com o tempo: são substituídos por pilhas mais novas; os padrões, no entanto, permanecem os mesmos. O objetivo é mostrar como as coisas funcionam e por que elas funcionam da maneira que funcionam.

Este livro conecta diferentes áreas comumente isoladas em material educacional: desenvolvimento de API, integração de banco de dados, segurança de aplicativos e processos de implantação. Ele oferece uma visão integral de como as maiores empresas de software criam software capaz de atender a milhões de usuários. Essa visão beneficia pessoas que estão iniciando suas carreiras como desenvolvedores de software e aquelas que passaram a maior parte de suas carreiras trabalhando em apenas uma dessas áreas.

Este livro possui 12 capítulos ordenados cronologicamente para acompanhar as fases de um projeto de desenvolvimento de software. Idealmente, você pode levar este livro com você e ler cada capítulo na ordem em que seu projeto evolui. Cada capítulo se baseia no anterior, por isso é aconselhável que você leia do início ao fim. No entanto, cada capítulo também é autocontido o suficiente para você retornar a qualquer um deles quando precisar atualizar seu conhecimento.

A primeira parte do livro cobre os aspectos essenciais do desenvolvimento de back-end. Ele descreve o processo de conversão de necessidades de negócios em requisitos, definindo APIs elegantes que são flexíveis o suficiente para evoluir com o aplicativo e escolhendo o tipo correto de banco de dados. Esta primeira parte também oferece um mergulho profundo no funcionamento interno de aplicativos baseados na web, construindo um modelo mental robusto que permitirá que você compreenda completamente as abstrações construídas sobre eles.

A segunda parte abrange padrões e processos necessários para construir qualidade em aplicativos de software: testes, segurança de aplicativos, gerenciamento de erros e logs, adoção de estrutura e integração e implantação contínuas. Esses conceitos são a base para o trabalho diário dos desenvolvedores de software em todo o mundo.

*A terceira e última parte do livro serve a dois propósitos:*Descreva como saltar de aplicativos simples para sistemas e conceitos distribuídos de grande escala que você pode usar para avançar em sua carreira e se tornar um desenvolvedor sênior.

Capítulo 1abrange os tópicos de resolução de problemas e levantamento de requisitos. Ele fornece informações sobre a importância dos aplicativos de software como ferramentas para resolver problemas. Este capítulo fornece heurísticas úteis para coletar com êxito os requisitos de negócios em um processo iterativo. Em seguida, o capítulo fornece uma visão de alto nível de aplicativos de software de

grande escala e todos os componentes e ferramentas que abordam os requisitos funcionais e não funcionais do projeto.

Capítulo 2é um mergulho profundo em aplicativos baseados na web e na arquitetura cliente-servidor. Ele detalha o funcionamento interno dos servidores web, orientando você a criar seu próprio servidor web baseado em Java e compará-lo com servidores prontos para produção, como o Express do NodeJS. Essa visão detalhada ajuda você a entender a "mágica" por trás dessas ferramentas, que é uma habilidade crítica para se tornar um desenvolvedor sênior.

Capítulo 3desambigua o termo "API". Ele explica o que são APIs, por que são úteis para desenvolvedores de back-end e como elas se encaixam no processo de criação de um aplicativo. Em seguida, o capítulo o orienta sobre como projetar APIs flexíveis e orientadas a dados e como escolher as ferramentas certas para implementá-las: REST, GraphQL e gRPC.

Capítulo 4cobrirá o estado e o gerenciamento de dados, que inclui bancos de dados. Ele fornece informações sobre por que precisamos de bancos de dados, como modelar dados da vida real e como escolher o banco de dados adequado para suas necessidades de negócios. Este capítulo aborda as diferenças entre SQL e NoSQL e destaca as diferenças entre os produtos mais comuns para cada um, como bancos de dados relacionais, de documentos e gráficos.

capítulo 5explora a área de testes. Ele detalha o processo de teste manual e automatizado e destaca as diferenças entre conceitos como mocks, stubs e duplos de teste. Este capítulo também fornece boas práticas para escrever testes unitários e de integração e suas diferenças. Em seguida, descreve outras áreas de testes não funcionais, como testes de desempenho e segurança.

Capítulo 6abrange a segurança do aplicativo. Este capítulo define os conceitos usados por especialistas em segurança de aplicativos que qualquer desenvolvedor de software deve conhecer. Ele descreve como integrar serviços de autenticação e autorização ao aplicativo, incluindo padrões do setor como OAuth2, SAML e OpenID Connect. Este capítulo também aborda algumas das vulnerabilidades de segurança mais comuns e aconselha como evitá-las.

Capítulo 7explica o tópico de gerenciamento de erros e logs em aplicativos de software. Ele destaca como algumas linguagens de codificação lidam com erros, como encontrar erros em aplicativos implantados em ambientes de produção e como centralizar e monitorar erros em aplicativos distribuídos usando ferramentas como Logstash, Elastisearch e Kibana.

Capítulo 8abrange um mergulho profundo em estruturas de aplicativos. O capítulo explica o que são estruturas, para que são usadas e os padrões que usam para resolver os desafios diários dos desenvolvedores de back-end. Em seguida, apresenta estruturas populares como Spring e Hibernate de Java e Django de Python. Ele contrasta como padrões como o MVC são implementados nesses frameworks e

fornece a base para os desenvolvedores escolherem o framework adequado para seu caso de uso.

Capítulo 9descreve como implantar um aplicativo em um ambiente de produção. Ele fornece algumas informações históricas sobre como os aplicativos foram entregues aos usuários, os desafios enfrentados e como os fluxos modernos de CI/CD (integração e implantação contínuas) permitem que as equipes de desenvolvimento entreguem seus aplicativos rapidamente. Também exploramos o conceito de ambientes replicáveis por meio da virtualização usando máquinas virtuais e contêineres Docker e como eles se integram ao processo de CI/CD.

Capítulo 10apresenta tópicos avançados para a criação de aplicativos distribuídos em grande escala. O capítulo aborda como medir o desempenho para encontrar oportunidades de melhoria. Em seguida, exploramos técnicas para melhorar o desempenho como cache, arquiteturas assíncronas e programação assíncrona através dos conceitos de "Promises" e "Futures". Apresenta algumas ferramentas utilizadas para aumentar o desempenho em grandes aplicativos como Redis e Kafka.

Capítulo 11dá um passo atrás e integra todos os capítulos anteriores em um processo para projetar um aplicativo de software. O capítulo descreve uma abordagem passo a passo para converter requisitos em especificações técnicas, estimar requisitos de tamanho de servidor e armazenamento e princípios usados para dimensionar aplicativos. Esse processo é um modelo especialmente útil para entrevistas de projeto de sistema.

Capítulo 12o último capítulo deste livro descreve conselhos de carreira para desenvolvedores de back-end que desejam se tornar desenvolvedores seniores. Ele descreve as responsabilidades típicas de desenvolvedores juniores e seniores e fornece diretrizes para se preparar para entrevistas técnicas, encontrar mentores e encontrar recursos para continuar aumentando seu conhecimento técnico.

Pacote de códigos e imagens coloridas

Acesse o link para baixar o
Pacote de códigoe as Imagens Coloridas do livro:

https://rebrand.ly/sueh55f

O pacote de código para o livro também está hospedado no GitHub em**https://github.com/bpbpublications/Backend-Developer-in-30-Days**. Caso haja uma atualização no código, ela será atualizada no repositório GitHub existente.

Temos pacotes de código do nosso rico catálogo de livros e vídeos disponíveis em**https://github.com/bpbpublications**. Confira-os!

CAPÍTULO 1

Como criar aplicativos multiusuário

Resolver problemas é o trabalho principal de um desenvolvedor de software; dedicamos nossas carreiras a adquirir as habilidades necessárias para sermos solucionadores de problemas eficientes. Aprendemos linguagens de computador, frameworks, bibliotecas e padrões de design para construir um conjunto de ferramentas diversificado para lidar com a grande variedade de problemas que este mundo oferece.

Neste capítulo, falaremos sobre resolução de problemas: discutiremos como os problemas que os seres humanos encontraram ao longo da história evoluíram. Veremos que os avanços tecnológicos são cíclicos, e entender esses padrões nos ajudará a prever os problemas que podemos encontrar no futuro e projetar melhores soluções.

Também discutiremos que tipo de problemas podemos corrigir hoje e como podemos traduzir declarações de problemas em desafios técnicos para os quais podemos oferecer soluções baseadas em software.

Em seguida, passaremos a capturar efetivamente todos os requisitos para um novo projeto, descrevendo alguns obstáculos que podemos encontrar no processo e boas práticas para resolvê-los.

Por fim, teremos uma visão de alto nível da variedade de ferramentas disponíveis para construir aplicativos de software. Em capítulos posteriores, exploraremos essas ferramentas em detalhes, mas usaremos este capítulo primeiro para entender como todas elas se encaixam.

Estrutura

Neste capítulo, aprenderemos os seguintes tópicos:

- Transformação digital e um pouco de história
  - Transformação digital e internet
- Projetando aplicativos para resolver problemas do mundo real
  - Cuidando dos problemas do usuário como desenvolvedor back-end
  - Encontrar um problema para resolver
- Definindo um caso de uso de amostra: O sistema de pedidos do Pizza Place.
- Definição de requisitos funcionais e não funcionais
  - Uma maneira ineficaz de coletar requisitos

    - O ciclo de definição de requisitos
- O design moderno do sistema: uma visão de dez mil pés

Objetivo

Depois de concluir este capítulo, você deve entender claramente o objetivo principal dos desenvolvedores de software: Resolver problemas do mundo real.

Ao final deste capítulo:

- Você entenderá como encontrar bons problemas para resolver como desenvolvedor.
- Você perceberá que devemos encontrar um problema para resolver antes de tentar construir um aplicativo.
- Você terá uma lista de heurísticas para definir com eficiência os requisitos funcionais e não funcionais e saber quais práticas evitar.
- Você entenderá as ferramentas disponíveis para nós para construir aplicativos modernos.

Transformação digital e um pouco de história

Durante a segunda metade do século XVIII, iniciou-se uma transformação na Europa: os produtos artesanais passaram a ser fabricados por máquinas. Movidas a vapor e mais tarde à eletricidade, essas máquinas permitiram que as pessoas produzissem mercadorias mais rapidamente e com menos dinheiro; com custos de produção mais baixos, os fabricantes poderiam alcançar mais consumidores do que nunca.

Esse período, conhecido como Revolução Industrial, mudou a forma como os negócios operavam e transformou profundamente a sociedade. As comunidades rurais começaram a crescer em espaços urbanos. Cidades geograficamente distantes aumentaram a comunicação e o comércio graças aos avanços no transporte (como a criação e melhoria de trens e barcos a vapor). Pequenas comunidades começaram a se fundir em maiores.

As pessoas se mudaram para as cidades à medida que muitos empregos agrários foram substituídos por trabalhos industriais. A mudança nunca é fácil, mas foi uma transição especificamente difícil para muitas pessoas cujas habilidades atuais nem sempre eram valiosas para o novo mercado de trabalho. Os artesãos que não conseguiam competir com as grandes máquinas de produção tiveram que se adaptar se quisessem continuar sustentando suas famílias.

Os empregos tornaram-se mais especializados e os trabalhadores precisavam de instrução para operar as novas máquinas, então novas escolas e cursos de aprendizagem foram criados. Com as pessoas concentradas nas comunidades urbanas, a educação também foi racionalizada. A educação formal e as profissões tornaram-se onipresentes.

Para acelerar tudo, o dinheiro também se concentrou em cidades onde os donos das fábricas agora podiam produzir mais por menos. A concentração nos recursos levou à desigualdade por parte de tut também permitiu mais investimentos em avanços tecnológicos, o que levou ao aperfeiçoamento das mesmas máquinas que iniciaram essa transformação.

Uma Segunda Revolução Industrial trouxe a produção em massa por meio da eletricidade, e uma Terceira Revolução Industrial trouxe a automação por meio da eletrônica e da tecnologia da informação. Cada uma dessas novas fases foi construída em cima da anterior, cada uma com um alcance mais amplo. A industrialização virou digitalização.

As pessoas foram obrigadas a se adaptar em cada uma dessas revoluções, nem sempre sem resistência.

Nota: Como aconteceu então, as pessoas no presente têm o mesmo tipo de medo de mudança e incerteza. As pessoas temem que algumas profissões se tornem obsoletas devido aos avanços na Inteligência Artificial, resultando na perda de empregos.

O termo "ludita" foi criado para descrever pessoas que não gostavam de novas tecnologias e suas ameaças a seus interesses, como segurança no trabalho e privacidade.

Muitos zombam de pessoas que não gostam dos avanços tecnológicos, mas raramente tentamos ter empatia. As revoluções industrial e digital não foram perfeitas nem socialmente justas. Ainda há muito debate sobre como a revolução industrial melhorou a qualidade de vida das pessoas além das aparentes riquezas que deu aos poucos proprietários de tecidos e seus círculos sociais.

Alguns estudos antropológicos descobriram que a desigualdade cresceu no Reino Unido durante a revolução industrial, pois "a industrialização precoce beneficiou desproporcionalmente mais o capital do que outros fatores de produção":

https://www.econstor.eu/bitstream/10419/201834/1/1671601041.pdf

Essa mesma conversa sobre tecnologia e desigualdade está acontecendo hoje, quando CEOs e fundadores de empresas de tecnologia são as pessoas mais ricas do mundo.

Espera-se que se sinta ansioso com o progresso tecnológico. Compreender como as pessoas foram historicamente capazes de se adaptar às novas dinâmicas sociais emergentes nos dá uma visão de como preparar a sociedade para o futuro.

Como o grupo que constantemente introduz mudanças, faz parte do nosso trabalho garantir que façamos tudo ao nosso alcance para garantir que os

avanços tecnológicos levem em consideração questões como desigualdade ou privacidade.

À medida que a era digital começou e a tecnologia se tornou mais do que apenas máquinas a vapor, a vida e as necessidades das pessoas também mudaram.

Transformação digital e internet

Suponha que tenha mapeado a quantidade de progresso tecnológico em função do tempo. Esse gráfico mostraria que o progresso tem sido exponencial desde o início da primeira revolução industrial. À medida que o tempo avança, damos saltos tecnológicos maiores.

Temos visto mais mudanças tecnológicas nas últimas duas décadas do que nos 200 anos anteriores, e estamos apenas começando. Agora vivemos em um mundo digital, e os avanços em campos como Machine Learning, IA e computação quântica estão apenas acelerando nossa adoção.

Dica: CPUs são uma boa referência de como a tecnologia avança. A Lei de Moore, mesmo quando estamos testando seus limites ultimamente, descreve como a capacidade dos processadores aumenta exponencialmente com o tempo. O avanço tecnológico progrediu como as CPUs: com o passar do tempo, o mundo muda em um ritmo mais rápido; descobertas acontecem com mais frequência.

Como nós, humanos, já vivemos essa transformação intensa e acelerada antes, podemos obter respostas para algumas perguntas que temos hoje. Por exemplo, muitas pessoas se perguntam o que acontecerá quando a IA começar a executar tarefas que as pessoas têm hoje. Ainda assim, como acabamos de ver, as pessoas sempre encontram uma maneira de se adaptar, mesmo ao custo de uma intensa mudança social e redistribuição de recursos.

Assim como nossas necessidades e como as atendemos mudaram durante as primeiras fases da Revolução Industrial, a criação e a ampla adoção da Internet transformaram a humanidade: a Internet é a plataforma sobre a qual a Terceira e Quarta Revoluções Industriais são construídas.

As necessidades das pessoas são agora mais complexas e abstratas. Por exemplo, as necessidades sociais vão além da interação com os pares em suas comunidades diretas; agora fazemos parte de grupos sociais não limitados pela distância ou pelo tempo. As pessoas esperam estar conectadas, mesmo morando em um país ou continente diferente.

Toda uma geração de pessoas agora prefere pedir comida online do que pedir pelo telefone ou simplesmente entrar em um restaurante. Pagamos nossas contas online. Novos filmes e programas de TV estão sendo estreados on-line em vez de por meio de cinemas ou assinaturas a cabo. Podcasts são agora tão onipresentes quanto o rádio. O streaming de música deslocou o consumo de CDs.

A tecnologia ainda está mudando os papéis das pessoas na sociedade e, como desenvolvedores de software, precisamos entender e aproveitar esse conhecimento. Assim como quando o agricultor teve que aprender a operar máquinas de fabricação, as pessoas que criam e transmitem conteúdo digital (por exemplo, YouTubers) são empregos que estão substituindo cada vez mais as profissões tradicionais (muitas delas destinadas a serem consumidas pela IA).

Os desenvolvedores de software que entenderam as novas necessidades de nossa sociedade em constante mudança construíram as ferramentas nas quais esses novos profissionais operam - ganhando milhões no processo.

Transformação de software

O campo de desenvolvimento de software é tudo menos estático. Poucos desenvolvedores de software trabalham com a mesma tecnologia em todas as suas carreiras. Adaptar-se e aprender novas tecnologias é um requisito essencial para a nossa profissão. Os melhores desenvolvedores de software entendem a mudança e se adaptam a ela.

A própria Internet mudou. A web deixou de ser a coleção de conteúdo estático unidirecional que era; agora é uma plataforma altamente dinâmica que permite a comunicação entre os proprietários dos sites e os usuários. Os aplicativos modernos fornecem valor com base nas informações que recebem de seus usuários.

Nossos próprios usuários mudaram: a Internet não é o lugar apenas para os técnicos, os acadêmicos ou os hobistas que já foi. Graças a anos de trabalho árduo de pessoas brilhantes e especialistas em interação Humano-Computador, a Internet é agora um espaço democrático onde você não precisa entender como o software é construído para usá-lo. Os usuários agora são avós, profissionais de qualquer área de especialização e até crianças pequenas.

A escala de software aumentou devido à expansão do uso da Internet. Os servidores da Web agora têm tantos usuários que um único computador poderoso não é suficiente para atender às solicitações de todos eles. Os dados que os aplicativos modernos consomem não cabem em uma única máquina, mesmo considerando os discos rígidos de alta capacidade que conseguimos desenvolver nos últimos anos.

O processo de desenvolvimento de software também continua mudando: não criamos mais código apenas para computadores; criamos aplicativos que agora rodam em smartphones, TVs e até torradeiras ou geladeiras. E você não precisa de um Ph.D. ou até mesmo uma educação universitária para construir código: os adolescentes estão ganhando milhões de dólares vendendo aplicativos que eles criam em seu tempo livre.

Porém, uma coisa não muda: as pessoas ainda têm problemas que precisam de soluções. Mesmo com os avanços que acabamos de descrever, milhões ainda não conseguem os serviços e recursos essenciais de que precisam para sobreviver. A parcela da humanidade que pode tirar proveito da tecnologia resolveu suas

necessidades básicas tão bem que tem tempo e recursos para se preocupar com outros problemas mais abstratos.

Projetando aplicativos para resolver problemas do mundo real

Muitos desenvolvedores de software esquecem um princípio simples, mas crítico: aplicativos são ferramentas. Eles são um meio para um fim, mesmo que o fim em si seja muito abstrato ou frívolo. E na maioria das vezes, o software em si não é esse fim.

Os desenvolvedores de software adoram criar aplicativos, e esse amor nos faz esquecer que, na maioria das vezes, criamos aplicativos para alguém que não somos nós mesmos.

A aceleradora de startups Y Combinator ofereceu uma série de palestras na Stanford University em 2014 intituladas “How to Start a Startup”. Nessas palestras, surgiu uma questão: Como escolher o tema ou ideia em que você poderia basear sua empresa. A solução proposta: Encontre um problema que seus usuários possam ter e encontre uma solução; não importa quão grande ou pequeno seja o problema, desde que seja um problema do mundo real.

Muitas startups falham porque se concentram mais na tecnologia que estão usando do que em criar algo que ajude as pessoas. Eles gastam muito dinheiro e tempo obtendo a pilha de tecnologia mais quente, os frameworks mais complexos. Para essas empresas falidas, todas essas camadas de complexidade escondem uma única verdade: não há problema real a ser corrigido, nenhum usuário que precise de seus produtos.

Outras empresas podem ter um problema real para resolver, mas ainda assim erram o alvo. Eles tentarão novamente construir algo complexo, inteligente demais para seu próprio bem. Eles poderiam ter construído uma solução mais direta com tecnologia menos “legal” por consideravelmente menos dinheiro e ainda oferecer soluções para seus usuários.

Seus usuários não se importam se você usou Java ou Python para criar o aplicativo de que precisam. As únicas pessoas que se importam com essas coisas são outros desenvolvedores, que não são seu alvo principal na maioria das vezes. Sem um problema real para resolver, os aplicativos nada mais são do que exercícios mentais divertidos para os desenvolvedores que os criam.

E não estamos dizendo que você não pode construir código só porque gosta dele. Mas quando você está sendo pago por outra pessoa para fazer isso, criar um aplicativo que funcione corretamente é uma prioridade.

Dica Termo de pesquisa: codificação do criativo.

Existe um tipo de programação de computador que não se baseia na resolução de problemas: a codificação criativa. Essa codificação não funcional é focada na criação de representações abstratas como trabalhos artísticos, visuais ou áudio.

Empresas como o Google criaram eventos para as pessoas construírem código como forma de expressão artística, e é um campo que nós, como desenvolvedores de software, estamos apenas começando a explorar. Muitos desenvolvedores front-end usam CSS para criar trabalhos visuais no navegador que ultrapassam os limites da tecnologia da web.

Assim como você pintaria um quadro ou escreveria uma música para se divertir e se expressar, considere criar código para mais do que começar seu próprio negócio ou ganhar dinheiro.

Cuidando dos problemas do usuário como desenvolvedor back-end

Os desenvolvedores de back-end trabalham em uma camada que é abstraída dos usuários: os usuários não técnicos terão dificuldade em ver a conexão direta entre seu trabalho e o aplicativo que eles usam e adoram.

Essa abstração é normal, pois é comum que desenvolvedores de back-end trabalhem em tarefas menos centradas no usuário: lidar com servidores, APIs e conectar-se a bancos de dados. Desenvolvedores frontend, por outro lado, tendem a trabalhar com requisitos mais centrados no usuário porque seu principal objetivo é construir a interface com a qual os usuários trabalharão diretamente.

Essa desconexão desafia os desenvolvedores de back-end: é fácil perder de vista os problemas que estamos tentando corrigir com nosso aplicativo. Você pode passar meses ou até anos em sua carreira sem saber como ou se o aplicativo em que está trabalhando atende às soluções certas de que seus clientes precisam.

É por isso que colocamos tanta ênfase na solução de problemas no início deste livro, pois ela deve ser nosso princípio orientador como desenvolvedores de software. Você será um engenheiro de back-end consideravelmente mais bem-sucedido se não esquecer seus usuários, mesmo que eles não usem diretamente o código que você cria.

Encontrar um problema para resolver

As chances são de que em algum momento de sua carreira como desenvolvedor back-end, você terá que trabalhar com sua equipe para encontrar um novo projeto para trabalhar. Pode ser encontrar uma missão para sua nova startup ou iniciar um novo projeto dentro de sua empresa existente. Concentrar-se na solução de problemas levará a uma atribuição mais otimizada dos recursos de sua equipe.

A maioria dos problemas que encontramos gira em torno de duas possibilidades: problemas que ainda não têm solução e problemas que têm solução, mas não são bons o suficiente ou funcionam apenas para um pequeno grupo de pessoas.

Tentar encontrar problemas sem uma solução conhecida é complicado: é da natureza humana tentar consertar coisas que não funcionam, então os problemas são resolvidos relativamente cedo ou os problemas permanecem sem solução porque as soluções são complexas ou inexistentes. Se você tiver sorte de encontrar uma solução para um problema que não foi resolvido antes, vá em frente e trabalhe nele; mas sinta-se à vontade para trabalhar em problemas que podem ser resolvidos melhor.

Nota: Problemas "menores" tendem a ser frequentemente ignorados pelos desenvolvedores. Nós nos concentramos em encontrar grandes problemas porque achamos que isso aumentará nossas chances de sucesso; quanto mais pessoas usarem a solução que construímos para elas, mais reconhecimento e receita teremos. No entanto, pensar em termos de problemas "grandes" ou "pequenos" não é a abordagem ideal.

Em sua palestra, "Competition is for Losers" Peter Thiel fala sobre como as Startups que se concentram em nichos de mercado têm mais chances de sucesso apenas porque têm menos concorrência. Limitar o escopo do seu projeto para resolver apenas um problema simples (e resolvê-lo bem) leva a ter mais tempo e recursos para explorar diferentes soluções.

Não devemos supor que as soluções existentes para problemas conhecidos sejam sempre a melhor abordagem. Os problemas tendem a mudar mais rapidamente do que as soluções, então as implementações ficam desatualizadas. Use esse conhecimento para manter sua mente e olhos abertos para soluções novas e melhores.

A diversidade é um ponto significativo a ser considerado ao encontrar problemas para resolver. Um problema comum com aplicativos ou soluções existentes é que a solução que eles fornecem funciona apenas para um grupo específico de usuários.

Por exemplo, veja aplicativos de edição de vídeo ou imagem. Aplicativos como Adobe Premiere ou Adobe Photoshop existem há muitos anos, mas seu alvo sempre foram usuários experientes que possuem um conhecimento muito especializado da área. Hoje em dia, graças aos smartphones e aplicativos que você pode obter gratuitamente, qualquer pessoa pode editar fotos e vídeos com qualidade profissional.

Muitos desenvolvedores de software assumem que todos os seus usuários fazem parte do mesmo grupo demográfico. Historicamente, o software sempre foi construído primeiro para usuários experientes. Empresas inovadoras como a Apple reconheceram a importância de construir produtos para não especialistas; eles devem seu sucesso a essa democratização da tecnologia.

Há um fato simples, mas poderoso: você não é seu usuário.

Você é o especialista em cada aplicativo que você escreve: será evidente para você como usar cada recurso porque você os escreveu e testou diariamente. Então, quando um usuário chega até você e diz que não sabe como usá-lo, pode ser um pouco frustrante, pois é algo óbvio para você.

É um equívoco comum pensar que a solução que funciona para você funcionará para todos. Mesmo quando você faz parte do grupo de usuários que seu aplicativo segmenta, sua opinião sobre como o aplicativo deve funcionar é inerentemente tendenciosa. Por isso é essencial conversar diretamente com seus usuários.

Quanto mais você entender a diversidade de sua base de usuários, mais fácil será para você encontrar seus desafios e adaptar melhor as soluções para eles.

Defina um exemplo de caso de uso: O sistema de pedidos do Pizza Place.

No início da carreira de qualquer desenvolvedor de back-end, é normal se sentir sobrecarregado pela falta de experiência em projetos do mundo real. Nesse ponto, a maioria dos projetos em que trabalhamos são:

- Projetos para escola ou acampamento de codificação, onde você pode pular várias etapas do processo de desenvolvimento de software devido a restrições de tempo.
- Projetos existentes aos quais você se juntou muito tempo depois que os principais projetos técnicos e decisões foram tomadas.

Não é até mais tarde em sua carreira que você terá a oportunidade de trabalhar em um projeto desde o início, encontrando um problema para resolver e tomando decisões com base nas necessidades do seu usuário.

Por meio deste livro, visitaremos várias técnicas para criar o back-end para aplicativos de nível de produção e as conectaremos a uma narrativa que permitirá que você obtenha experiência em áreas anteriormente inexploradas para você. Para isso, vamos começar com o problema a ser resolvido.

Há uma pequena pizzaria chamada “The Pizza Place” no seu bairro. Eles têm apenas alguns funcionários hoje, mas o gerente é uma senhora brilhante com uma visão de longo prazo: construir ótimas pizzas com ingredientes de qualidade a preços acessíveis. Ela está preocupada que a pizza que as crianças locais estão comprando não seja saudável o suficiente para eles, então ela quer oferecer pizzas cozidas com ingredientes limpos e frescos.

O Pizza Place já oferece refeições no local, mas quer começar a receber pedidos online. A gerente não quer perder uma comissão pesada para um serviço de pedidos de comida online existente, então ela e seu marido decidem que a solução é investir na construção de seu aplicativo de pedidos personalizado. Sim, será caro no começo, mas depois de executar os números, eles economizarão muito dinheiro a longo prazo.

Você e dois de seus colegas acabaram de fundar uma nova startup focada em ajudar as empresas locais a ingressar na nova era digital: você é o principal desenvolvedor de back-end e os outros dois são um desenvolvedor/designer de front-end e um vendedor.

A proprietária do Pizza Place vê um dos anúncios que você postou on-line e realmente acredita que você pode ajudá-la a realizar o sonho de levar uma ótima pizza para todos. Ela entra em contato com você e sua equipe para construir seu sistema de pedidos. Todos vocês agendam uma ligação para passar os detalhes.

Definição de requisitos funcionais e não funcionais

É sabido que um dos primeiros passos em um novo projeto é reunir os requisitos do usuário. Muitas vezes dividimos os requisitos em dois:

- **Requisitos funcionais ou de negócios**: Esses são os objetivos principais do aplicativo, as coisas com as quais seus clientes se preocupam: o problema principal e a solução abstrata para ele.
- **requisitos não Funcionais**: Esta é a definição dos elementos do sistema necessários para implementar a solução proposta para os requisitos funcionais. Isso abrange quantos servidores você precisará, que tipo de banco de dados etc.

Supõe-se que você tenha lido várias vezes sobre as diferenças entre esses requisitos, portanto, não iremos mais longe nesses detalhes. O importante é que ter uma lista detalhada de requisitos nos ajudará a escolher as peças certas para construir o melhor aplicativo que pudermos.

Dica: Modelo mental: Definir requisitos funciona de maneira muito semelhante ao conceito matemático de programação linear: você tem uma equação objetiva que precisa otimizar encontrando valores de variáveis que otimizam uma função de destino, mas há uma combinação infinita de valores que você pode usar.

Ao considerar um conjunto de restrições (múltiplas igualdades ou desigualdades), podemos descartar todos os valores de variáveis que não contribuem para obter os melhores resultados para nossa equação objetiva.

No nosso caso, os requisitos do sistema são as restrições que nos permitem delimitar o escopo do problema que estamos tentando corrigir. Podemos encontrar a combinação ideal de ferramentas como bancos de dados ou bibliotecas de código observando a área delimitada por nossos requisitos.

Embora a matemática seja uma habilidade comum para engenheiros de software, não se preocupe se você não estiver familiarizado com programação linear, pois não a usaremos neste livro. Este é apenas um exemplo para ajudar a

construir um modelo mental mais robusto para quem já presenciou esse tipo de problema.

Muitas vezes falamos sobre requisitos, mas não como defini-los de forma eficaz. Muitos desenvolvedores esperam que os clientes definam seus próprios requisitos funcionais com perguntas equivocadas como “o que você quer que o aplicativo faça?”.

Alguns clientes têm uma boa ideia do que querem, mas a maioria não. É seu trabalho como desenvolvedor de software segurar suas mãos nesta parte do processo para encontrar a melhor solução para eles.

Uma maneira ineficaz de coletar requisitos

Um bom ponto de partida para entender o processo de coleta de requisitos é um exemplo real. Vamos discutir a experiência de um desenvolvedor (que pode ou não ser o autor deste livro) e as falhas que ele teve no início de sua carreira. Não há melhor oportunidade de aprendizado do que cometer muitos erros.

Alguns anos na carreira desse desenvolvedor, ele supervisionou a coleta dos requisitos do sistema para um cliente em potencial. Ele entrou em contato com o gerente da empresa do cliente para agendar algumas reuniões para definir os requisitos. Ele agendou três sessões por “muita cautela”. Para ter certeza de que ele acertou todos os requisitos.

Durante as reuniões, ele perguntou ao gerente o que ele queria que o aplicativo fizesse. O gerente falou sobre como o negócio funcionava, e o desenvolvedor e sua equipe fizeram todas as perguntas que puderam pensar. Eles tomaram muitas notas e desenharam muitos diagramas. Ao final da coleta de requisitos de negócios de três dias, eles se sentiram como um especialista no assunto do negócio do cliente.

Avanço rápido algumas semanas depois; eles começam a codificar o aplicativo. À medida que construíam a interface do usuário definida em seus diagramas, eles começaram a ter muitas dúvidas. Suas perguntas variavam de não saber o que o aplicativo deveria fazer em casos específicos até ter entendido completamente mal o conteúdo de um relatório gerado dinamicamente que o aplicativo deveria renderizar.

Como esse desenvolvedor deveria dizer a seu gerente que eles precisavam agendar mais reuniões com o cliente? Ele havia estabelecido a expectativa de que essas reuniões eram tudo o que eles precisavam. O gerente desse desenvolvedor teve que conversar com o cliente para solicitar mais reuniões para tirar suas dúvidas, o que atrasou nosso projeto.

O desenvolvedor aprendeu que precisava definir as expectativas certas: sempre haverá mais perguntas.

O ciclo de definição de requisitos

A definição dos requisitos do sistema é um processo iterativo, muitas vezes intercalado com o próprio design da solução.

Não existe uma receita mágica para encontrar todos os requisitos para um novo projeto. No entanto, aqui estão algumas heurísticas que você pode usar para guiar seu caminho:

1. Agende uma ou duas reuniões com a seguinte agenda:
   - Descubra qual é o problema que seu cliente está tentando resolver.
   - Obtenha uma visão detalhada do negócio: metas de curto, médio e longo prazo e restrições orçamentárias.
2. Solicite a ajuda de um dos especialistas de domínio do seu cliente.
3. Agende várias sessões de trabalho com o especialista do domínio com os seguintes objetivos e ações:
   - Saiba como o processo existente funciona hoje.
   - Entenda como seu cliente está atualmente trabalhando em torno do problema.
   - Construir protótipos e revisar requisitos.
4. Com a ajuda do especialista no assunto, itere várias vezes entre o design do protótipo e a definição e esclarecimento dos requisitos.

Descubra qual é o problema que seu cliente está tentando resolver

Como já discutimos muitas vezes neste capítulo, a primeira coisa que você precisa saber é qual problema está tentando resolver. Se este é um projeto que você mesmo definiu, você já deve conhecer o problema, e esta etapa já está quase concluída.

No entanto, novos clientes terão seus próprios casos de negócios, e é por isso que solicitam seus serviços. Os casos de uso comuns incluem:

- Construindo um aplicativo para automatizar um processo existente. O objetivo aqui é reduzir os custos substituindo os processos manuais existentes que são propensos a erros ou demorados.
- Crie um novo aplicativo para agregar valor aos processos de negócios existentes. O objetivo é criar novas funcionalidades cujo objetivo, por sua vez, seja resolver os problemas de nossos clientes; coisas como oferecer um novo serviço ou produto online.
- Refatorar um aplicativo existente. O objetivo é substituir (totalmente ou apenas partes) um aplicativo legado que não esteja operando conforme o esperado ou cuja manutenção seja muito cara.

Seu cliente em potencial pode não ter uma ideia clara de qual solução ele precisa (é por isso que ele pagará a você), mas saberá os desafios que está enfrentando. Compreender o que os levou a você em primeiro lugar é o primeiro passo.

Obtenha uma imagem detalhada do negócio

Depois de saber o que aflige seu novo cliente, o próximo passo é entender quem ele é e seus objetivos a curto, médio e longo prazo.

Por que precisamos entender qual é o objetivo de longo prazo deles? Não podemos apenas focar no aplicativo que eles querem construir hoje? Você pode. É possível limitar o seu conhecimento sobre o seu cliente a este problema específico e ainda assim ter um projeto de sucesso.

Mas, ao obter uma visão mais profunda do negócio do seu cliente, você obtém duas coisas:

- Uma visão melhor de como o aplicativo pode crescer e casos de uso futuros.
- A possibilidade de encontrar outras áreas onde você pode ajudar seus clientes significa mais negócios para você.

O primeiro é especialmente importante do ponto de vista técnico: permite que você escolha melhor as ferramentas que não se tornarão obstáculos quando surgirem requisitos futuros.

Por exemplo, eles podem estar usando um provedor de serviços de cobrança específico hoje, mas planejam mudar para um provedor diferente alguns anos depois. Esse conhecimento informará que você deve ter cuidado para não acoplar fortemente a implementação do sistema ao provedor existente.

Dica: termo de pesquisa: à prova de futuro

Um termo comumenteusado em projetos de software é “à prova de futuro”, o que significa criar recursos em seu aplicativo que não são necessários hoje, mas podem ser usados no futuro.

A prova de futuro geralmente é considerada uma prática ruim, pois introduz esforço desperdiçado e complexidade desnecessária em sua base de código ao implementar recursos que talvez nunca sejam usados. Código morto é sempre dívida de tecnologia.

No entanto, construir aplicativos enquanto entende quais podem ser os requisitos no futuro não é à prova de futuro. Criamos aplicativos para os casos de uso de hoje usando ferramentas que não criarão dívida tecnológica para os possíveis casos de uso de amanhã.

Uma coisa sobre o negócio que é fundamental para entender é os limites que eles têm no orçamento. Não é o mesmo que criar um aplicativo para uma empresa multinacional de bilhões de usuários com orçamentos milionários do que criar um aplicativo para uma empresa local com apenas algumas centenas de usuários.

As restrições orçamentárias fornecerão muitas informações sobre requisitos não funcionais. Se eles tiverem um orçamento limitado, seu cliente pode aceitar fazer alguma troca entre o número de servidores que você precisa e o número de usuários que eles podem ter. Ferramentas de código aberto podem ser preferidas a produtos licenciados.

Seu cliente pode não saber como criar um orçamento para este projeto; podemos orientá-los na criação de um se entendermos completamente seus objetivos, os requisitos funcionais e não funcionais e como mapeá-los em infraestrutura e ferramentas.

Solicite a ajuda de um dos especialistas em domínio do seu cliente

Antes de terminar essas primeiras reuniões com seu novo cliente, peça que ele atribua um especialista de domínio que entenda o problema a ser corrigido. Pode ser a pessoa que deve usar o novo aplicativo ou que trabalha com o processo existente.

Como você trabalhará em estreita colaboração com esse especialista de domínio, defina as expectativas certas: você precisa de alguém com quem possa entrar em contato facilmente para obter respostas para suas perguntas. Alguém que possa treiná-lo efetivamente em como essa parte do negócio funciona.

Essa parte é essencial, pois você não quer cometer os mesmos erros que eu: perder muito tempo com gerentes ou diretrizes na fase inicial de exploração. Ou acreditando que você tem conhecimento suficiente para ser o especialista do domínio (99% das vezes, você não será o especialista nesse domínio).

Saiba como o processo existente funciona hoje

A maneira mais eficaz de coletar requisitos é ver como os clientes operam hoje. Com a ajuda de seu especialista de domínio, comece a explorar os detalhes das operações comerciais. Se possível, peça permissão para acompanhá-los enquanto fazem seu trabalho diário.

Especialistas de domínio não são necessariamente usuários especializados. Se você estiver criando um aplicativo de exercícios, um especialista em domínio pode ser alguém que se exercita com frequência ou um usuário que o usaria para ficar em forma.

Ao observar o trabalho de seu especialista de domínio, você aprenderá como o sistema funciona hoje fazendo as seguintes perguntas:

- Como é um dia normal para a pessoa que deve usar o aplicativo?
- Em que contexto eles devem usá-lo? Em um escritório? Na Academia?
- O que o especialista gosta no processo atual? O que eles não gostam?

Ao ver o trabalho do especialista em domínio, observe se ele faz algum "hack" ou "truque" para facilitar seu trabalho. Esses podem ser requisitos ocultos diretamente relacionados ao problema principal.

Lembre-se de um fato importante: você não pode melhorar um processo sobre o qual não sabe nada. Você não se tornará um especialista trabalhando neste projeto, mas contar com um especialista de domínio lhe dará proficiência suficiente para encontrar todos os requisitos.

Construir protótipos e revisar requisitos

Como mencionado anteriormente, a coleta de requisitos é iterativa. Depois de saber como o novo aplicativo deve funcionar, é hora de trabalhar nos primeiros protótipos.

Dica: termo de pesquisa: protótipos de baixa fidelidade e protótipos de alta fidelidade

A construção de protótipos faz parte de uma fase exploratória na construção de aplicativos. É uma tarefa que geralmente fica nas mãos do designer da equipe ou dos engenheiros de front-end, mas qualquer um pode construir protótipos eficazes.

Protótipos de baixa fidelidade são esboços construídos grosseiramente que apenas destacam a funcionalidade crítica. Eles não incluem detalhes como cores, fontes, imagens ou mesmo texto, pois isso distrai do ponto que você deseja discutir.

Protótipos de baixa fidelidade são boas ferramentas para usar no início de um projeto para explorar ideias e encontrar mal-entendidos nos requisitos.

Protótipos de baixa fidelidade são baratos, pois podem ser desenhados em quadros brancos, notas adesivas ou até guardanapos que você pode ter enquanto toma café.

Os protótipos de alta fidelidade são complexos e variam de esboços de alta definição a aplicativos totalmente funcionais. Eles são melhores para mais tarde no projeto validar suposições sobre interação do usuário, design visual e conteúdo de texto.

Os protótipos de alta fidelidade podem ser caros, pois exigem tempo de um designer ou desenvolvedor de software para serem criados. Use-os com parcimônia.

Ao criar protótipos de baixa fidelidade à medida que reúne os requisitos, você pode confirmar se suas suposições sobre o aplicativo estão corretas. Encontrar lacunas em seu modelo mental desde o início economizará muito esforço e dinheiro a longo prazo.

Nesta etapa, você deseja envolver seus desenvolvedores front-end e seus designers, pois eles podem colaborar na criação do protótipo e obter respostas para perguntas que possam ter por conta própria.

Não gaste muito tempo ou esforço criando esses protótipos, pois eles provavelmente mudarão ao longo do caminho. Você só deseja incluir as coisas essenciais necessárias para impulsionar ainda mais a conversa sobre os requisitos.

*Figura 1.1*mostra um exemplo de um protótipo de baixa fidelidade. A baixa qualidade muda o foco para o caso de uso em que um usuário seleciona um item no menu e clica no botão Pedir:

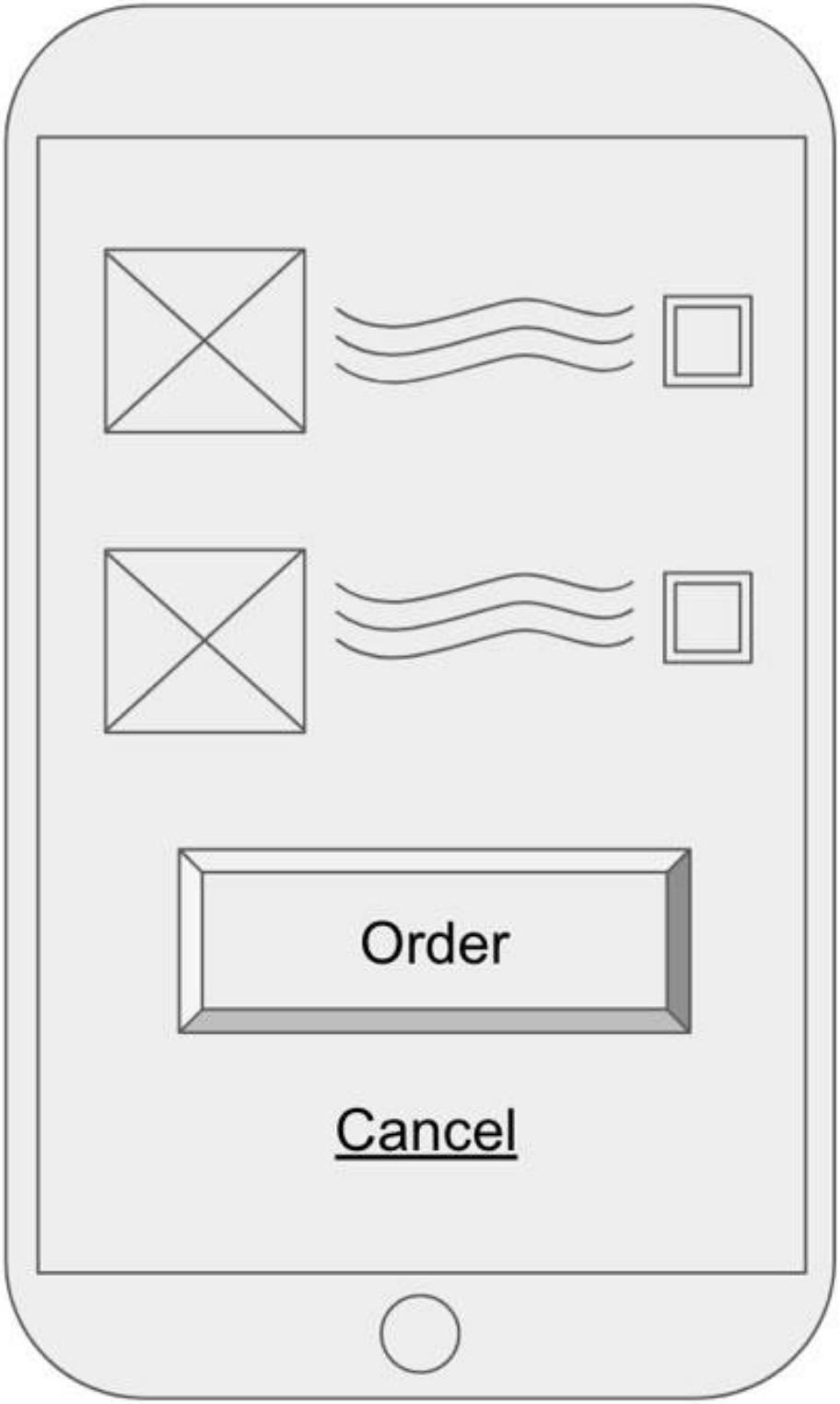
Order
Cancel

Figura 1.1: Um exemplo de um protótipo de baixa fidelidade.

Revise seus protótipos com o especialista do domínio. Eles vão validar suas suposições ou dizer o que você errou. A vantagem dos protótipos de baixa fidelidade é que eles podem ser atualizados com relativa rapidez. Usando esse feedback, itere para ver como o especialista de domínio opera e encontre a suposição correta.

Conforme você obtém o acordo nos requisitos definidos, crie um documento central para incluir todos os fatos do projeto.

Caso de uso: Definindo requisitos para o Pizza Place

Depois de passar o dia todo no The Pizza Place, descobrimos que eles querem uma operação simples para a primeira versão do aplicativo: exibir um menu para seus usuários em seus computadores ou smartphones e permitir que eles peçam comida.

Eles estão apenas começando com esse esforço de modernização, então eles querem a opção mais barata disponível para atender sua carga de clientes atual de uma média de 100 pedidos por dia. Antes de sair, o gerente lhe diz que, se o sistema funcionar bem, eles vão querer expandi-lo à medida que recebem mais pedidos diários.

Essas informações sobre seus objetivos e orçamento nos dão uma visão clara: a Pizza Place não precisa de um sistema distribuído com clusters de servidores hoje, mas está conquistando clientes rapidamente. Não demorará muito para que o aplicativo precise ser dimensionado, e nós levaremos em conta isso.

Depois de trabalhar com o desenvolvedor front-end e os cozinheiros, você apresenta uma coleção de protótipos e requisitos ao proprietário da empresa.

Ela acha que algumas de suas sugestões têm recursos de que não precisam no curto prazo (como permitir que os usuários façam muitas personalizações nas coberturas de suas pizzas). O Pizza Place oferece apenas um menu simples de pizzas com coberturas de alta qualidade pré-selecionadas, portanto, um módulo de personalização é um trabalho extra que eles não precisam.

Graças ao trabalho de alguns dias com a equipe do Pizza Place, podemos definir um documento com todos os requisitos e premissas. Usamos esse documento como a fonte da verdade e recebemos a aprovação do cliente.

Tendo entendido as necessidades do nosso novo cliente, é hora de vermos um menu de todas as diferentes ferramentas que temos para satisfazer esses requisitos.

O design moderno do sistema: uma visão de dez mil pés

Como agora sabemos que encontrar soluções para os problemas de nossos usuários tem prioridade sobre a escolha de uma pilha de tecnologia, vejamos a variedade de

ferramentas atualmente disponíveis para desenvolvedores de software em cada camada do processo de desenvolvimento de aplicativos.

A seguir está uma visão de alto nível dos elementos mais comuns usados na construção de aplicativos modernos:

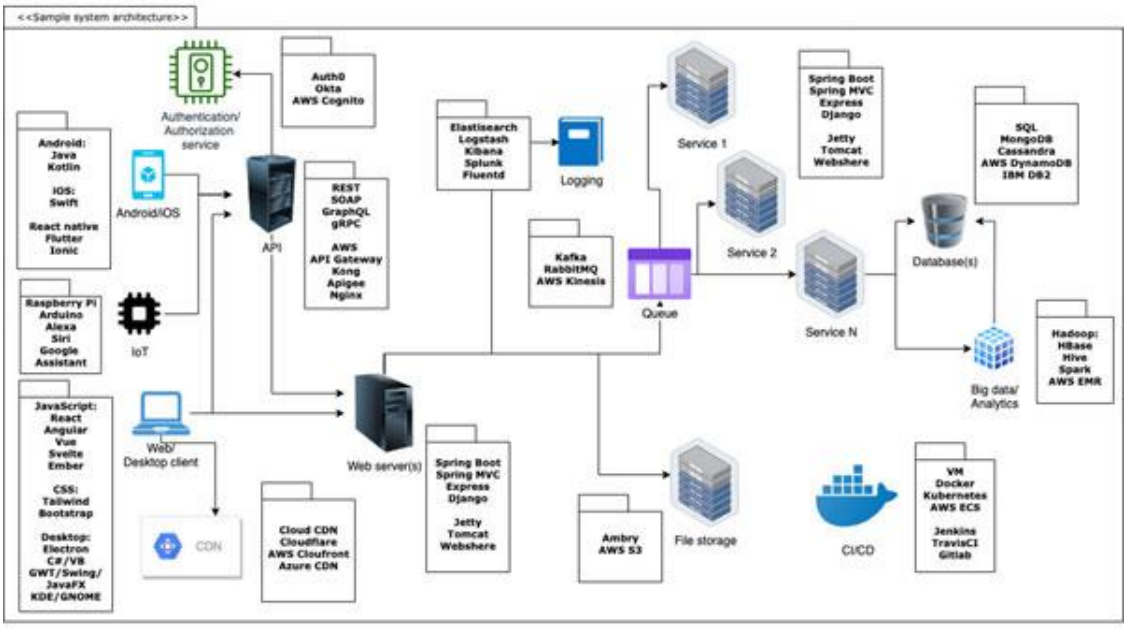

Figura 1.2: Arquitetura do sistema de exemplo

Esmagador, não é? Temos várias camadas, como front ou back-end, cada uma com vários serviços, como armazenamento de dados, lógica de negócios e gerenciamento de infraestrutura; cada um tem diferentes provedores e implementações, com seus prós e contras.

Tirando o front-end do caminho

Observe que não estamos fazendo uma distinção rígida na imagem entre um front-end e o back-end. Este limite flexível tem duas razões:

Primeiro, esta é uma visão geral de todo o sistema, de ponta a ponta.

Em segundo lugar, no desenvolvimento moderno, a linha divisória entre as responsabilidades dos desenvolvedores front-end e back-end tornou-se confusa, especialmente em startups onde os orçamentos e as equipes são pequenos e onde a maioria dos engenheiros de software acaba trabalhando em toda a pilha.

É comum que desenvolvedores de front-end trabalhem na API ou que desenvolvedores de back-end atualizem o código do cliente para habilitar os recursos de que precisam.

Essa fusão de responsabilidades tornou-se tão onipresente que levou ao surgimento do título indescritível de "desenvolvedor full-stack", que nada mais é do que um engenheiro de software com experiência em tecnologia de ambas as extremidades da pilha.

Vamos nos concentrar no front-end por um segundo. Podemos ver que esta arquitetura de exemplo oferece suporte a um conjunto diversificado de clientes:

- Clientes da Web (navegadores como Chrome, Firefox, etc.)
- Aplicativos móveis nativos (Android, iOS)
- **Internet das Coisas**(IoT) com tecnologia integrada, como placas Raspberry Pi ou Arduino.
- Dispositivos "inteligentes" como Amazon Alexa ou Google Assistant
- Clientes de desktop nativos

Historicamente, antes do uso generalizado de dispositivos inteligentes como telefones e tablets, as aplicações web tradicionais tinham um forte acoplamento com os serviços de back-end. Os aplicativos da Web dinâmicos dependeriam do mesmo servidor de back-end para fazer várias coisas:

- Buscar dados do banco de dados
- Executar lógica de negócios
- Transforme e formate dados em HTML
- Retorne o HTML gerado dinamicamente para o navegador

No entanto, a necessidade de reutilizar a lógica de negócios tornou-se crítica à medida que uma ampla gama de modernos ganhou popularidade. O fluxo de execução das aplicações web foi dividido em dois:

- Uma API para publicar lógica específica de negócios (que também inclui busca de dados e transformação de dados)
- Um servidor Web para renderizar o front-end, que chama a API de negócios, se necessário
- Aplicativos móveis que fazem chamadas para a mesma API que o aplicativo da web

Essa separação de interesses ajudou a separar o desenvolvimento de front-end e back-end: desde que um contrato seja totalmente estabelecido no nível da API, o back-end e o front-end podem ser desenvolvidos em paralelo. Vamos aprofundar essas preocupações de design no capítulo sobre (*Capítulo 4, Projetando APIs*).

Com essa separação clara entre o front-end e o back-end, vamos retirar o front-end da imagem e simplificá-lo um pouco, para que possamos nos concentrar em todas as áreas que discutiremos neste livro.

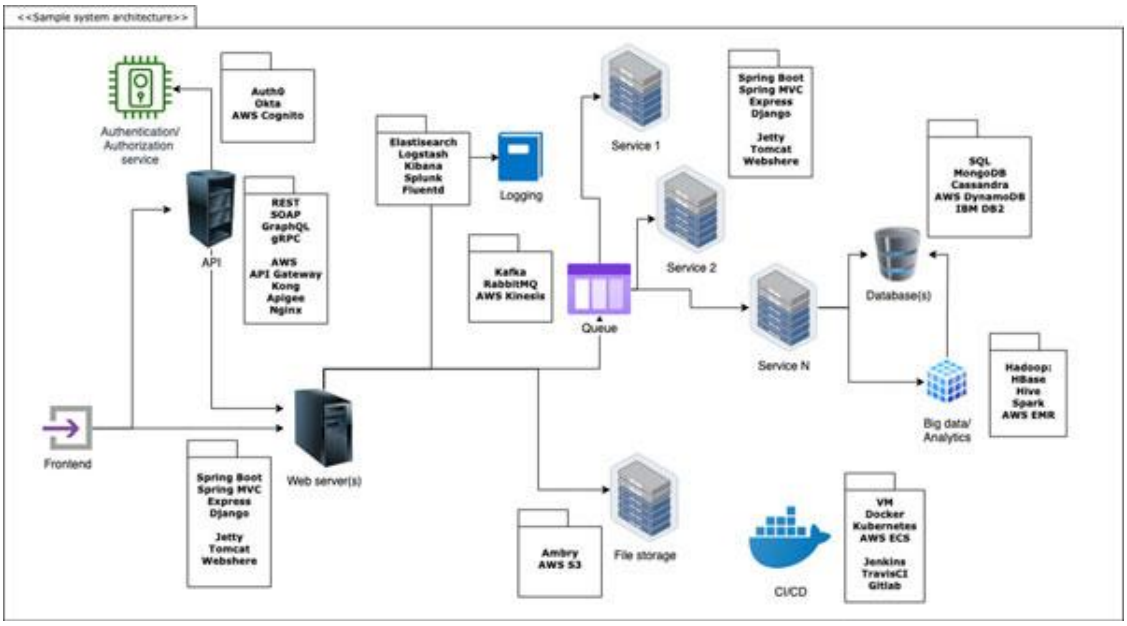

Figura 1.3: Arquitetura do sistema de exemplo

Não se preocupe se você se sentir um pouco perdido. Você não deve conhecer cada parte deste diagrama, nem deve memorizá-lo. Esta é apenas uma prévia de todos os tópicos que discutiremos neste livro.

Cada componente exibido em*Figura 1.3*é um tipo diferente de ferramenta em nosso cinto. Todos eles atingem objetivos diferentes, mas apenas um subconjunto deles é necessário para o aplicativo médio. Por exemplo, talvez você não precise de um serviço de armazenamento de arquivos se não estiver lidando com arquivos binários em seu aplicativo.

Blocos de construção

Servidores da websão responsáveis por receber solicitações HTTP dos clientes diretamente ou de uma camada de API externa. Pode ser apenas um servidor se não houver muito tráfego ou pode ser um conjunto de vários servidores atrás de um Load Balancer. Veremos mais detalhes em*Capítulo 2, A Arquitetura Cliente-Servidor*.

A camada de API recebe solicitações de clientes ou outros aplicativos. Esta pode ser uma camada lógica (serviços rodando no servidor web) ou semi-física (serviços independentes rodando em seus próprios servidores) que cuida de tarefas como autorizar solicitações de clientes usando um serviço de autorização. A camada de API pode até limitar solicitações ou cobrar dos usuários por usá-la. Veremos mais detalhes em*Capítulo 4, Projetando APIs*.

Para sistemas distribuídos, é comum usar uma fila. Permite a comunicação assíncrona entre serviços. As filas fornecem resiliência ao sistema caso serviços individuais

estejam inativos. Obteremos mais informações sobre isso em*Capítulo 11, Criando aplicativos de alto desempenho*.

Exploração madeireiraserviços fornecem armazenamento de informações de operação. Erros, avisos ou métricas de eventos são armazenados aqui para análise posterior. Logging é um tópico que será discutido em*Capítulo 8, Lidando com Erros*.

O aplicativo pode ser dividido em um ou vários serviços de negócios. Estes também podem ser lógicos ou semi-físicos. Toda a segunda parte deste livro é dedicada a como construir esses serviços.

Base de dadosservices fornecem um local para manter os dados do nosso aplicativo. Serviços como armazenamento de arquivos são tipos especializados de armazenamento de dados.*Capítulo 5, Gerenciamento de dados de ponta a ponta*,se aprofunda sobre como armazenamos e gerenciamos dados.

Integração Contínua e Entrega Contínua(CI/CD) é uma metodologia focada na construção da infraestrutura necessária para compilar e implantar nosso aplicativo. Dentro*Capítulo 10, Implantando aplicativos*, discutimos como aproveitar conceitos como controle de versão, máquinas virtuais (VMs) ou contêineres para implantar nosso aplicativo à medida que fazemos alterações nele.

Existem muitos outros blocos de construção que não incluímos aqui. Essa omissão não se deve a esses componentes serem menos críticos, mas porque aumentaria o escopo deste livro.

Além disso, observe que não estamos nos referindo a vários desses blocos por tecnologias específicas (por exemplo, estamos usando "armazenamento de dados" em vez de MySQL, MongoDB, etc.). Ao manter esses conceitos no nível certo de abstração, veremos padrões comuns que essas implementações específicas compartilham, permitindo-nos dominá-los com relativa facilidade.

Compreender os componentes listados lhe dará conhecimento suficiente para ser um desenvolvedor de back-end proficiente.

Nos capítulos seguintes, dissecaremos cada componente desses diagramas juntamente com as boas práticas para usá-los na criação de aplicativos de qualidade.

Conclusão

A maioria dos aplicativos que construiremos em nossas carreiras são ferramentas para corrigir problemas para as pessoas. Se o aplicativo não fornecer a solução certa, os usuários o abandonarão.

Ter essa visão orientada para a solução de problemas como desenvolvedor de back-end lhe dará uma vantagem sobre outros desenvolvedores: ela orientará todas as suas decisões técnicas, tornando-o um desenvolvedor mais eficiente e bem-sucedido; mais

clientes vão querer usar o código que você cria, e mais empresas vão querer que você faça parte de suas equipes porque seu código faz as coisas acontecerem.

Encontrar problemas para resolver não é uma tarefa simples, mas se você for proativo o suficiente e fizer as perguntas certas, encontrará coisas para trabalhar que farão a diferença. Fornecer as soluções certas para problemas simples é o caminho para o sucesso das maiores empresas de tecnologia.

Seguindo um processo iterativo, guiado pelo feedback do especialista de domínio do seu cliente, você poderá identificar todos os requisitos do sistema necessários para escolher os componentes certos para sua aplicação.

As aplicações modernas possuem um ecossistema de componentes e ferramentas em constante evolução. Mais do que aprender as especificações de cada ferramenta, devemos nos concentrar em aprender padrões de projeto que abstraem os benefícios de todo o conjunto de ferramentas.

No capítulo seguinte, vamos mergulhar profundamente no tipo de aplicativo mais onipresente que construiremos como desenvolvedores de back-end: aplicativos da Web.

Perguntas

- De todos os aplicativos instalados em seu telefone, escolha um que facilite sua vida E resolva apenas um ou dois problemas. Por que uma solução tão específica causa impacto em você?
- Você já baixou um aplicativo que desinstalou imediatamente porque não forneceu nenhum valor para você? Se sim, o que faltou?
- Pense em três empresas ou startups que você acha que são bem-sucedidas. Que problemas eles resolveram quando começaram? Eles se concentraram em uma área ou resolveram mais de um problema?

CAPÍTULO 2

A arquitetura cliente-servidor

Para se tornar um desenvolvedor de back-end proficiente, precisamos delimitar claramente quais são nossas responsabilidades. Essas responsabilidades são fortemente definidas pelos padrões de arquitetura que usamos ao construir nosso aplicativo.

O padrão de arquitetura mais proeminente que usamos no desenvolvimento de back-end é a arquitetura cliente-servidor. Não há praticamente nenhum aplicativo sendo construído atualmente que não dependa desse padrão, pelo menos em parte.

Esta é uma lição tão básica que muitos livros assumem que você sabe tudo sobre ela. É comum que essa suposição possa facilmente levar a mal-entendidos sobre como os aplicativos da Web funcionam; o mais comum é pensar que HTTP e cliente-servidor são um mapeamento 1:1. Descobriremos que cliente-servidor é muito mais amplo que HTTP.

Neste capítulo, discutiremos como a arquitetura cliente-servidor é implementada em aplicativos modernos baseados na Web, em mais de um nível de abstração.

Esta não é uma descrição detalhada de um protocolo específico (como HTTP), mesmo que incluamos detalhes sobre ele. Incluiremos apenas informações suficientes para que você tenha uma boa compreensão de como as coisas funcionam.

Estrutura

Neste capítulo, aprenderemos os seguintes tópicos:

- Detalhes da arquitetura
  - **Camada de abstração:**Cliente front-end e cliente back-end
  - **Camada de abstração:**Cliente de serviço de acesso a dados, servidor de banco de dados
- **HTTP:**A linguagem da web
- Implementando um servidor web
  - O principal processo
  - Servindo uma resposta
  - Suporte multiusuário com multi-threading
- Usando um servidor pronto para produção
  - Habilitando HTTPS no Express
- **Arquitetura em camadas:**Separando totalmente o cliente do servidor
  - Dividir clientes
- Computação cliente versus servidor
- Servidores Web como serviços sem estado
  - Armazenando dados da sessão
- **Caso de uso:**Aplicando uma arquitetura cliente-servidor ao aplicativo Pizza Place
- Cliente-servidor versus peer-to-peer

Objetivos

Ao final deste capítulo, você terá uma forte compreensão de como as tecnologias cliente-servidor funcionam nos bastidores e como podemos aproveitar esse conhecimento para nos tornarmos um desenvolvedor de back-end melhor.

Você também aprenderá sobre algumas das configurações críticas em um servidor web: Compressão de resposta e criptografia de comunicação por meio de HTTPS.

Detalhes da arquitetura

Ter capacidade de computação e armazenamento suficiente para realizar tarefas que hoje consideramos simples era proibitivamente caro há algumas décadas. A maioria das grandes instituições, como universidades ou governos, possuía computadores operacionais com recursos de computação de alto nível (ou pelo menos recursos suficientes para fornecer valor a seus usuários).

Historicamente, os primeiros exemplos de arquiteturas cliente-servidor são baseados na necessidade de acesso remoto a servidores de alta capacidade. Terminais 'burros' foram criados para permitir a operação desses computadores sem precisar estar fisicamente presente na mesma sala que eles. Esses foram os primeiros clientes e se comunicaram com seus servidores por meio de conexões diretas ou redes privadas.

À medida que a tecnologia avançava, os clientes se tornavam menos 'burros'. Agora, os clientes modernos têm mais poder de computação e capacidade de armazenamento do que a maioria dos supercomputadores das últimas décadas. No entanto, a arquitetura cliente-servidor não está indo a lugar nenhum.

Definido de forma simples, os servidores são equipamentos de alta capacidade de armazenamento e alto poder de computação que armazenam e processam a maioria dos dados relacionados a objetivos funcionais específicos.

Um cliente é um dispositivo menor - com menos recursos de computação - que é usado para acessar os recursos do servidor. Um servidor pode receber solicitações de N clientes, conforme mostrado na figura a seguir:

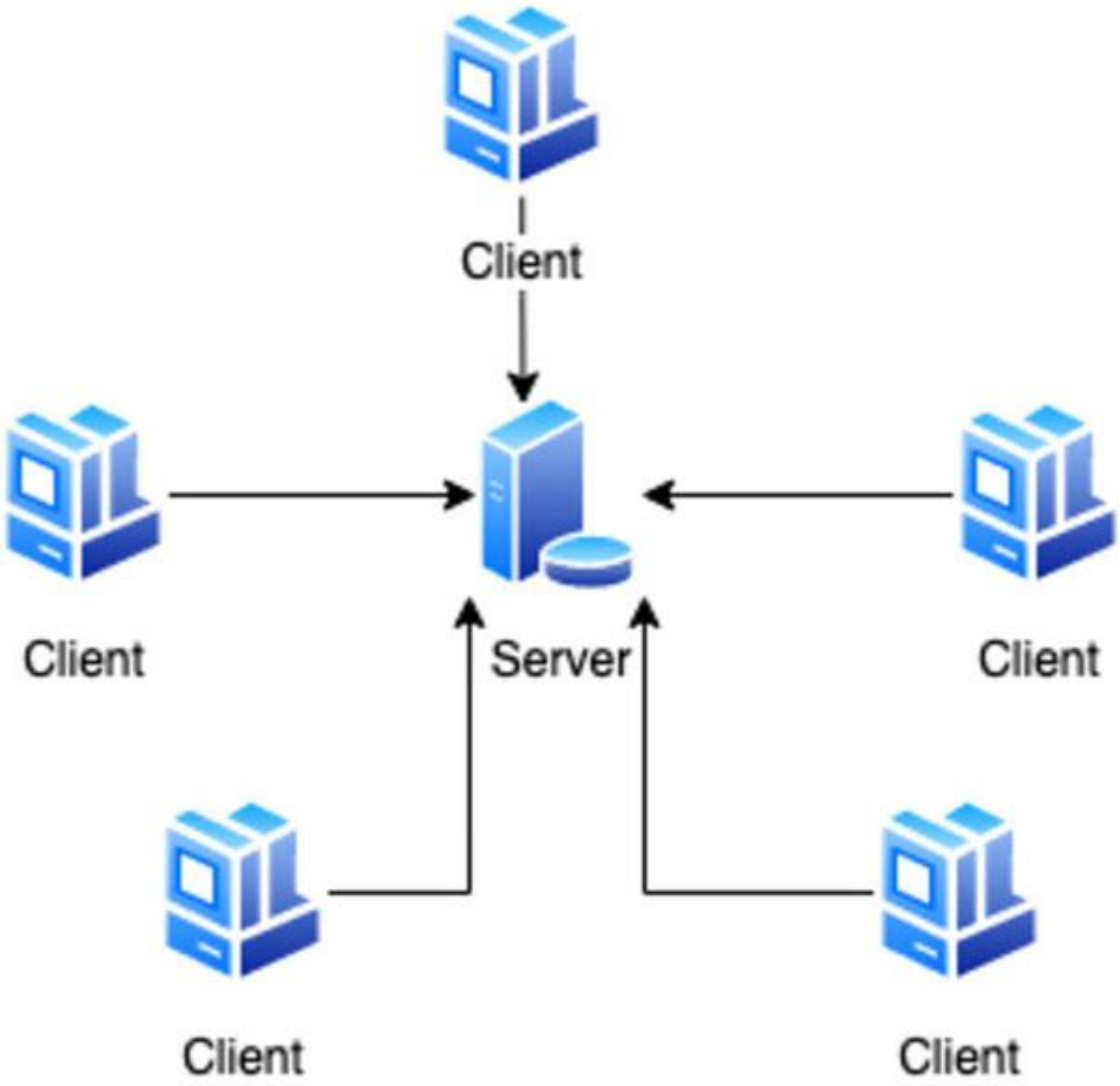

Figura 2.1: Arquitetura cliente-servidor simples. Vários clientes se conectam a um único servidor

Como visto no capítulo anterior, podemos criar clusters com vários servidores, cada um com funções diferentes. Mesmo se tivermos vários servidores, a definição de 'cliente-servidor' ainda é válida. Você pode pensar em toda a interação entre clientes e servidores como um dispositivo de computação único, abstrato e de grande escala. Todas as partes se juntam para se tornar uma única unidade “servidor”.

Um equívoco comum é acreditar que o cliente-servidor se aplica apenas à interface web, quando na realidade o cliente-servidor é usado como padrão em várias áreas do desenvolvimento de aplicativos.

A seguir estão alguns exemplos de áreas que implementam ativamente a arquitetura cliente-servidor:

- FTTP (para armazenamento de arquivos)

    - **Cliente:**Um programa que permite aos usuários fazer upload, download ou modificar arquivos hospedados em um servidor remoto.
    - **Servidor:**O servidor que armazena os arquivos.
- Cliente de serviço de acesso a dados, servidor de banco de dados
    - **Cliente:**Serviço de processamento de dados, com adaptadores de banco de dados como Open Database Connectivity (ODBC) ou Java Database Connectivity (JDBC)
    - **Servidor:**Servidor de banco de dados

Várias instâncias da arquitetura cliente-servidor podem fazer parte de um único aplicativo maior.

Camada de abstração: cliente front-end e cliente back-end

A instância mais comum da arquitetura cliente-servidor é a aplicação web. A função de cliente é atribuída às interfaces de usuário (aplicativos web, aplicativos móveis, dispositivos IoT e assim por diante) e a função de servidor é atribuída a todos os serviços para os quais as interfaces de usuário fazem solicitações, todos agrupados em uma única camada de abstração : o back-end.

É importante observar que o papel do cliente pode ser atribuído às interfaces do usuário, mas nem todos os clientes precisam envolver a interação direta do usuário. Existem clientes que, por sua vez, são usados por outros servidores (como clientes de banco de dados) para fazer solicitações a outros servidores. Pensar na interface do usuário e no cliente como os mesmos termos é um erro muito comum.

Camada de abstração: cliente de serviço de acesso a dados, servidor de banco de dados

Sistema de gerenciamento de banco de dados relacional(RDBMS) implementações como SQL são baseadas em uma arquitetura cliente-servidor: Um servidor de armazenamento de dados centralizado que é acessado por clientes leves.

Este modelo de armazenamento de dados concentra toda a informação num serviço central que tem a função de servidor. Em seguida, vários clientes podem enviar solicitações de 'leitura' e 'gravação' para nosso servidor de banco de dados.

Além dos óbvios ganhos de desempenho proporcionados por ter um servidor (ou cluster de servidores) inteiramente dedicado ao armazenamento de dados, também obtemos algumas garantias de consistência se assumirmos que o banco de dados é a fonte de verdade para a aplicação.

As coisas ficam um pouco complexas quando alguns bancos de dados implementam arquiteturas distintas do servidor cliente. Por exemplo, bancos de dados como o Cassandra têm uma distribuição interna de nós ponto a ponto em seus clusters. Pode

parecer que a arquitetura cliente-servidor não se aplica a esses tipos de banco de dados; mas do ponto de vista de usuários e serviços externos, o cluster de nós de mesmo nível ainda é um servidor por si só.

A principal conclusão aqui é que os conceitos de cliente e servidor são abstratos: eles podem ter várias instâncias cada e ainda ser considerados um único "cliente" ou um único "servidor".

HTTP: a linguagem da web

Discutimos a arquitetura cliente-servidor de um ponto de vista abstrato. Agora, vamos discutir o exemplo mais comum dessa arquitetura: HTTP.

Protocolo de Transferência de Hipertexto(HTTP) permite toda a comunicação entre aplicativos cliente (tanto desktop quanto baseados em dispositivos móveis) e os servidores web. As tecnologias de serviço da Web, como REST e SOAP, dependem internamente de HTTP.

Dentro*Figura 2.2*,podemos ver o diagrama de atividades para a comunicação entre um cliente solicitando uma página web index.html e fazendo o envio de um formulário por meio de uma solicitação POST:

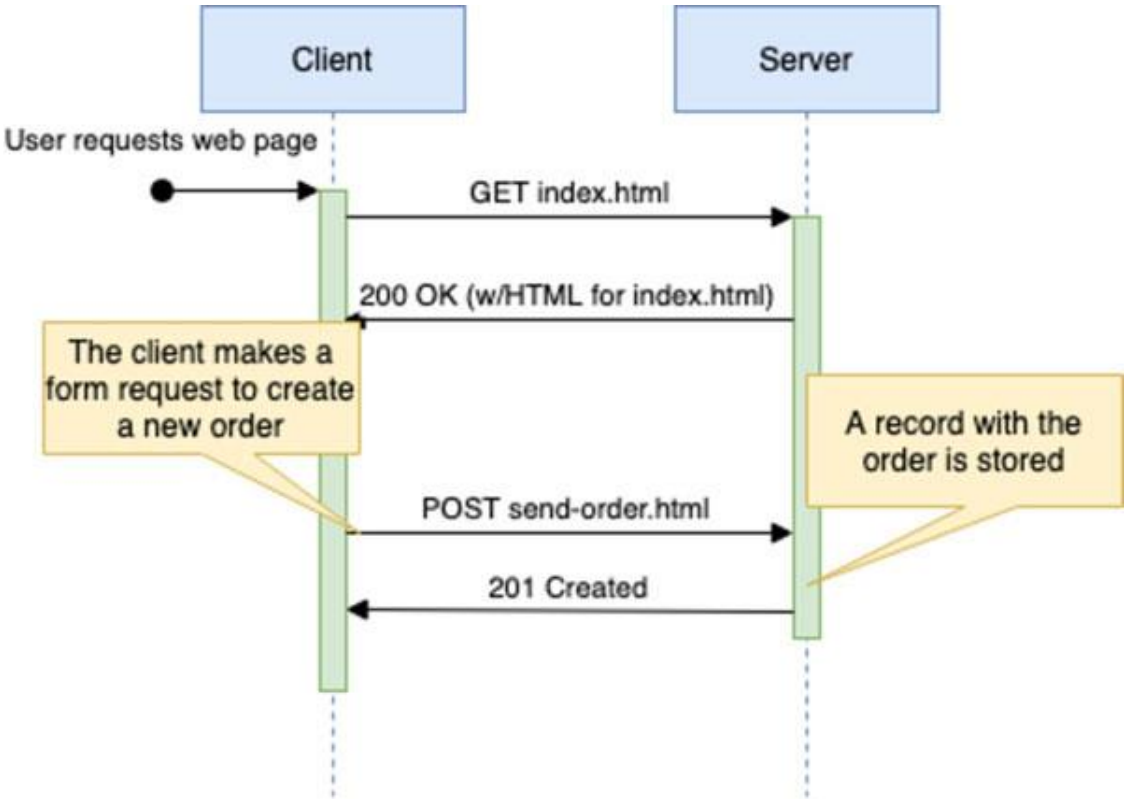

Figura 2.2: Exemplo de uma interação entre um cliente web e um servidor web usando HTTP

O HTTP é tão básico no desenvolvimento web que há chances de você já conhecer todos os conceitos básicos. Mas, ao mesmo tempo, o HTTP é composto de tantos conceitos abstratos que pode ser difícil relacioná-lo com casos do mundo real se você não tiver experiência suficiente.

Quando comecei minha carreira no desenvolvimento de back-end, havia uma grande lacuna na minha compreensão de como os servidores da Web se estendiam a partir de aplicativos mais básicos do tipo Hello-World. O código que construímos em linguagens como C++ ou Java na escola ou no início da minha carreira seguiu uma abordagem procedural. O código que você cria para aplicativos da Web, no entanto, usa um paradigma de programação orientado a eventos.

Como um aplicativo baseado em procedimentos se torna um aplicativo da Web orientado a eventos?

Dica Termos de pesquisa: programação procedural e programação orientada a eventos

Essas duas são abordagens muito diferentes de construção de código. **Dentro**programação procedural, os desenvolvedores descrevem o comportamento do programa uma linha de cada vez, um passo de cada vez, de ponta a ponta.

Dentroprogramação orientada a eventos, os desenvolvedores criam o código para reagir a determinados eventos: um usuário clica em um botão, um arquivo é baixado, uma resposta de rede é recebida etc.

A cola que se encarrega de traduzir o código entre os dois tipos de paradigmas de programação está escondida dentro da implementação do servidor web. Esta cola parece mágica para desenvolvedores inexperientes.

Se você tiver experiência em desenvolvimento da Web, poderá entender completamente como funcionam os aplicativos procedurais ou como funcionam os aplicativos orientados a eventos. Mas é provável que você não tenha visto o código que permite que ambos interoperem. Por isso, neste capítulo, vamos construir um servidor web muito simples.

A construção de um servidor tem dois objetivos: primeiro é examinar os aspectos internos do HTTP como um protocolo; o que é mais útil do que apenas recitar a especificação HTTP. Em segundo lugar, desmistifica o funcionamento interno dos servidores web; que nada mais são do que aplicativos regulares que também são compilados e executados (e procedimentais em sua execução). Não há nada realmente mágico dentro deles.

Implementando um servidor web

Os servidores da Web são peças complexas de software. Eles lidam com solicitações de vários usuários, lendo dados de soquetes e portas, formatando solicitações e respostas HTTP e assim por diante.

Muitos desenvolvedores passam anos em suas carreiras sem entender completamente como os servidores web funcionam internamente. Provavelmente, você nunca precisará construir seu próprio servidor web para um aplicativo pronto para produção; e você realmente não quer.

Entre construir seu próprio servidor web personalizado e usar um que esteja disponível comercialmente, você deve sempre escolher o servidor existente. Há muitas coisas a serem consideradas para garantir que um servidor esteja pronto para produção, então é melhor confiar em produtos que já foram amplamente testados antes.

Dito isso, vamos implementar um servidor web apenas para construir um modelo mental mais forte. Assim que você perceber que a operação básica de um servidor web é realmente muito simples, você se sentirá mais confiante em seu conhecimento sobre a construção de código backend para aplicações web.

Nota: Sobre modelos mentais

Exercícios que podem parecer um pouco desnecessários no início (como construir seu próprio servidor web quando você quase nunca vai querer usá-lo em um aplicativo de produção) são úteis para construir um modelo mental forte.

UMAmodelo mental é a nossa compreensão de como algo funciona. Os modelos mentais não se aplicam apenas ao software, pois você pode ter um modelo mental de como funciona um carro, como funciona sua cafeteira ou como funciona um processo burocrático.

Quando seu modelo mental sobre uma coisa é extremamente detalhado, você pode facilmente diagnosticar quando algo errado acontece dentro da própria coisa. Você pode entender seus pontos fortes e fracos e melhorias de design.

Se você quiser se tornar um especialista em qualquer campo, esforce-se para construir um modelo mental forte do funcionamento interno do que deseja estudar. Você deve ser quase capaz de imaginar cada parte em detalhes, como um mapa ou esquema.

Primeiro, construiremos uma versão de thread único do servidor. Em seguida, com algumas etapas simples, estenderemos o exemplo para usar um pool de encadeamentos para permitir várias solicitações simultâneas.

Usaremos Java para este exemplo, importando apenas bibliotecas de base. Sem frameworks ou produtos externos. Se você não estiver familiarizado com Java, tenha certeza de que não há muitos códigos específicos de Java e quase todas as linguagens possuem bibliotecas de rede que permitem que você faça isso.

O principal processo

De uma visão simplificada e de alto nível, um servidor web é um loop infinito que lê constantemente o soquete de um servidor. Por exemplo, expomos o socket em localhost ou 127.0.0.1, e uma porta TPC específica que não é usada atualmente pelo servidor, neste caso, a porta 8080.

Nota: Anatomia de um servidor: TCP/IP, portas e soquetes

TCP/IP é umprotocolo de rede usado para trocar informações. É usado pelo HTTP para direcionar solicitações e respostas de e para um servidor web.

Os servidores da Web podem ser identificados por seu endereço IP (e pelo nome de domínio que resolve através do DNS para esse IP, por exemplo, google.com). Vários serviços podem ser expostos no mesmo servidor, cada um usando uma porta específica. A combinação de um endereço IP e uma porta é conhecida comosoquete.

As portas sãonumerados e os primeiros 1024 são atribuídos a serviços conhecidos. Por exemplo, HTTP geralmente é roteado pela porta 80 e HTTPS pela porta 443.

As portas acima de 1024 não são pré-atribuídas e podem ser usadas pelo seu aplicativo para expor serviços personalizados. Por exemplo, durante o desenvolvimento, uma porta comum usada para HTTP é 8080, mas pode ser facilmente exposta usando qualquer outra porta como 8081, 8082, etc.

Localhost

Quando umservice começa a ser executado em um servidor, geralmente é acessado por seu endereço IP público e uma porta específica.

No entanto, você pode chamar um serviço de dentro do mesmo servidor em que ele está sendo executado usando o endereço IP "loopback" em vez de seu endereço público. O endereço de loopback em um servidor é sempre 127.0.0.1 e o alias do DNS é 'localhost'.

Portanto, se você estiver dentro de um servidor de produção executando um aplicativo da Web como example.com, poderá fazer uma solicitação para localhost:80 e deverá ver exatamente a mesma página da Web.

Enquanto o processo principal continuar em loop, o servidor estará ativo. Vejamos o processo principal:

```
1. import java.io.*;
2. importar java.net.ServerSocket;
3. importar java.net.Socket;
```

```
classe pública Servidor {
final privado estático int PORT = 8080;
public static void main(String[] args) lança Exception {
tente (ServerSocket serverSocket = new ServerSocket(PORT)) {
enquanto (verdadeiro) {
tentar {
Cliente de soquete = serverSocket.accept();
handleClient(cliente);
}
catch(erro de exceção) {
err.printStackTrace();
}
}
}
}
}
```

Em cada iteração, o processo principal verificará o soquete para ver se uma solicitação foi feita. Assim que o cliente fizer uma solicitação, serverSocket.accept() retornará uma instância de um cliente de soquete que conterá o fluxo de dados para uma solicitação do usuário.

É importante mencionar que serverSocket.accept() é uma operação de bloqueio. A execução chegará a essa linha de código e fará uma pausa até receber uma solicitação de entrada. Assim que a solicitação for recebida e processada, um novo loop acontecerá e (a menos que haja um erro) o servidor pausará novamente em serverSocket.accept() na próxima iteração.

Assim que uma solicitação for recebida, a função handleClient será chamada:

```
private static void handleClient(Socket client) lança IOException {
// Obtém o fluxo de entrada
BufferedReader br = new BufferedReader(
new InputStreamReader(client.getInputStream())
);
List<> requestLines = new ArrayList<>();
Linha de cordas;
// Lê todas as linhas da requisição até encontrar uma linha em
branco
Faz {
linha = br.readLine();
requestLines.add(linha);
} while(!line.isBlank());
//…
```

```
14. }
```

O cliente de soquete possui um método getInputStream, que retorna o conteúdo da solicitação. Este é um fluxo de bytes, mas como sabemos que é uma solicitação HTTP, podemos interpretar esses bytes como texto.

Podemos ler o fluxo de entrada do soquete usando um BufferedReader. O leitor tem métodos que nos permitem converter o fluxo de entrada em um iterador que, por sua vez, lê a solicitação uma linha por vez.

Nós iteramos pelas linhas de texto na solicitação até encontrarmos uma linha em branco que marca o final da solicitação.

Observação: BufferedReader tem métodos que retornam um Java Stream em vez de um iterador. Embora isso possa ser útil em certos casos, lembre-se de que o fluxo de entrada não será fechado. Ele permanecerá conectado até que o cliente navegue, feche o navegador ou expire a solicitação.

Nesses casos, os métodos de fluxo BufferedReader travarão até que o cliente feche a conexão, mesmo que uma linha em branco seja retornada no fluxo da resposta.

oA API Java Stream foi introduzida no Java 8 e, se você é um desenvolvedor Java e deseja aprender mais sobre ela, pode verificar o link 'Java Stream API' na seção de documentação para obter mais detalhes.

Neste ponto, você pode adicionar um ponto de interrupção do depurador dentro de handleClient, executar o programa, abrir uma janela do navegador e navegar até http://localhost:8080/index.html.

Quando a execução atinge o ponto de interrupção, você pode ver a solicitação HTTP na lista de strings que coletamos do fluxo de entrada:

```
GET /index.html HTTP/1.1

Anfitrião: localhost:8080

Conexão: manter vivo

Controle de cache: max-age=0

User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)
AppleWebKit/537.36 (KHTML, como Gecko) Chrome/92.0.4515.131 Safari/537.36
```

Você pode ver alguns valores ligeiramente diferentes ou cabeçalhos extras não incluídos neste exemplo, que definem coisas como cache, autenticação, gerenciamento de sessão etc. Isso pode depender do seu ambiente local, como o tipo de navegador usado para fazer a solicitação.

A primeira linha da solicitação segue um formato específico onde indicamos o verbo ou ação a ser executada, o destino da solicitação ou recurso ao qual aplicar essa ação e a versão HTTP.

Existem 9 tipos diferentes de verbos que podem ser usados; os mais comuns são GET e POST. Diferentes combinações de um verbo com o mesmo recurso podem indicar tipos distintos de operações.

Uma solicitação POST ou PUT seria semelhante à solicitação GET que acabamos de ver, com a adição de que a solicitação pode incluir um corpo abaixo de todos os cabeçalhos. Esse corpo pode ser dados binários, XML, JSON, etc.

Vamos visitar esses verbos com mais detalhes e ver como usá-los em*Capítulo 3, Projetando APIs*.

Servindo uma resposta

Tendo recebido e analisado uma solicitação do cliente, é hora de fazer algo com ela. Tenha em mente que, a partir do momento em que o servidor recebe a solicitação do cliente, o cliente aguardará uma resposta; então é melhor agirmos o mais rápido possível.

Apoiaremos as seguintes duas operações diferentes:

- **Retornar um arquivo HTML estático**: Isso emula o comportamento de servidores como Apache ou nginx que retornam arquivos estáticos para o cliente quando solicitados. Outros arquivos estáticos a serem veiculados podem ser arquivos JavaScript, CSS, imagem ou vídeo.
- **Retornar uma resposta HTML gerada dinamicamente**: Esta é uma forma comum de funcionamento de tecnologias de aplicativos da Web, como Java Servlets ou PHP. Eles constroem a resposta em tempo de execução com base nas informações fornecidas na solicitação.

Para dar suporte a esses dois casos em nosso exemplo, precisamos de uma condição para indicar ao servidor quando executar cada um. Apenas por causa deste exemplo, se o cliente solicitar /dynamic.html, retornaremos uma resposta de compilação dinâmica. Para quaisquer outros recursos, tentaremos buscar o arquivo estático solicitado (por exemplo, index.html). Se o arquivo não existir, retornaremos um erro 404 Not Found:

```
1. final privada estática String DATE_FORMAT_NOW = “aaaa-MM-dd
   HH:mm:ss”;
2. final privada estática String DYNAMIC = “/dynamic.html”;
3. byte estático privado final[] NOT_FOUND_HTML = “<h1>Não encontrado
   :(</h1>”.getBytes();
4. // …continuação de handleClient
5. // Analisa o caminho solicitado da primeira linha na solicitação
6. String[] requestLine = requestsLines.get(0).split(“ “);
7. Caminho da string = requestLine[1];
8. Caminho filePath = Paths.get(“.”, caminho);
```

```
// se o caminho solicitado for dynamic.html, imprime o dia de hoje
if(DYNAMIC.equals(caminho)) {
sendResponse(cliente, “200 OK”,
“texto/html”,
getDynamicResponse()
);
}
// senão, imprime arquivo estático
else if (Files.exists(filePath)) {
sendResponse(cliente, “200 OK”,
Files.probeContentType(filePath),
Files.readAllBytes(filePath)
);
// senão, imprime mensagem de erro
} senão {
sendResponse(cliente, “404 não encontrado”,
“texto/html”,
NOT_FOUND_HTML
);
}
}
byte estático privado[] getDynamicResponse() {
Calendar cal = Calendar.getInstance();
SimpleDateFormat sdf = new SimpleDateFormat(DATE_FORMAT_NOW);
String resposta = String.format(
“<h1>Resposta dinâmica</h1> Hoje é %s”, sdf.format(cal.getTime()));
return resposta.getBytes();
}
private static void sendResponse(Socket client, String status,
String contentType, byte[] content) throws IOException {
String LINE_BREAK = “\r\n”;
Saída OutputStream = client.getOutputStream();
output.write(("HTTP/1.1 " + status).getBytes());
output.write(("ContentType: " + contentType +
LINE_BREAK).getBytes());
output.write(LINE_BREAK.getBytes());
output.write(conteúdo);
output.write((LINE_BREAK + LINE_BREAK).getBytes());
output.flush();
cliente.fechar();
}
```

Dentro de handleClient, verificamos o recurso a ser buscado. Contamos com a estrutura VERB-RESOURCE-HTTP_VERSION que mencionamos anteriormente para extrair o nome do recurso. Se o recurso solicitado for "/dynamic.html", geraremos HTML dinamicamente para exibir a hora e a data atuais:

```
1. String resposta = String.format(
2. "<h1>Resposta dinâmica</h1> Hoje é %s", sdf.format(cal.getTime()));
```

Este pode ser um exemplo muito simples de HTML gerado dinamicamente, mas na prática, você provavelmente irá buscar dados de um banco de dados, convertê-los em um formato que possa ser entendido por seus usuários e depois convertê-los em HTML.

Para a rota de arquivo estático, criaremos um arquivo com o nome index.html e o colocaremos na mesma pasta onde nosso servidor será executado:

```
1. <corpo>
2. <h1>Resposta estática</h1>
3. Este é index.html
4. </body>
```

A função sendResponse cuida de formatar os cabeçalhos de resposta e renderizar o conteúdo no fluxo de saída do soquete. O fluxo lê arrays de bytes, então convertemos o texto que queremos retornar em nossa resposta em bytes (usando o método getBytes convenientemente incluído, parte da classe String).

Tendo criado os dois recursos, podemos acessá-los através de um navegador.

Abra uma nova janela do navegador e solicite http://localhost:8080/index.html para buscar a resposta estática. O navegador deve exibir uma mensagem de texto conforme exibido em*Figura 2.3*:

Figura 2.3:localhost:8080/index.html

De forma semelhante, podemos acessar o recurso que gera HTML dinamicamente abrindo outra janela do navegador e navegando até http://localhost:8080/dynamic.html. Você verá o mesmo texto mostrado em*Figura 2.4*:

Figura 2.4:localhost:8080/dynamic.html

Você pode imprimir a resposta diretamente no servidor ou usar as ferramentas de desenvolvedor do seu navegador para vê-la:

```
HTTP/1.1 200 OK

Acesso-Controle-Permitir-Origem: *

Conexão: Keep-Alive

Codificação de conteúdo: gzip

Tipo de conteúdo: texto/html; charset=utf-8

…



<html>

…
```

O formato da resposta HTTP é o seguinte: A versão HTTP, o código de status e um texto de status. O conteúdo real da resposta (neste caso, o HTML para index.html ou dynamic.html) é retornado abaixo dos cabeçalhos de resposta.